A Generic Linear Inequalities Solver (LIS) with an Application for Automated Air Traffic Control

Paul T. R. Wang*, William P. Niedringhaus, Matthew T. McMahon

The MITRE Corporation, Center for Advanced Aviation System Development (CAASD), 7515 Colshire Drive, McLean, Virginia, 22102-7539, USA

Abstract The authors illustrate the use of new algorithms for solving the feasibility problem of systems of linear inequalities in variables that define location, distance, speed, and time between aircraft in 4D space to achieve the desired aircraft movement obeying safety separation, speed limit, air traffic control restrictions, and desired flight path. This new linear inequalities solver (LIS) technique is applicable to any system of linear inequalities including linear programming, eigenvector programming, and systems of linear equalities. The three new LIS algorithms are: 1) Simplex alike method to reduce the worst infeasibility of a linear system, 2) a zero-crossing method to reduce the sum of all infeasibilities of a given linear system, and 3) a de-grouping algorithm to reduce the size of the set of constraints with the worst infeasibility. These set of new LIS algorithms were coded in Java and tested with MITRE CAASD’s airspace-Analyzer with hundreds of aircraft over thousands of variables, and tens of thousands of linear constraints. At high precision, solution of the LIS matches the solution obtained from the same LP problem using conventional Simplex method and methods used by Cplex, GNU LP solver, etc. The advantages and challenges of this generic LIS versus known, well-established algorithms for linear inequalities are also discussed.

Keywords Solution of Linear Inequalities, Algorithms, Air Traffic Control, Homogeneous Linear System, Self-dual Formulation, Linear Infeasibility, etc

1. Introduction

The authors present new concepts and algorithms to solve any system of linear inequalities in homogeneous form, which includes linear programs, eigenvectors, orthogonal space, and systems of linear equalities or inequalities. Four key algorithms were developed, coded, and tested to reduce the worst case of infeasibility, sum of infeasibilities, and the number of constraints with the worst infeasibilities. This paper is organized into the following 10 sections:

Section 2. Linear Systems Feasibility (LSF)
Section 3. An Airspace Analyzer using LP
Section 4. A Simplex alike method
Section 5. Zero-crossing in LSF
Section 6. De-grouping the worst infeasibility set
Section 7. Convergence/Completeness Analysis
Section 8. Decomposition and Combination of LIS
Section 9. LIS Scalability
Section 10. Advantages and Challenges
Section 11. Conclusion and Future Work

2. Linear Systems Feasibility (LSF)

Given any system of linear inequalities or equalities in vector and matrix form, \( \bar{A}\bar{x} \geq \bar{b} \) where \( \bar{A} \) is an \( m \) by \( n \) matrix, \( \bar{x} \) and \( \bar{b} \) are \( n \) by 1 column vectors.

We define the linear system feasibility (LSF) for \( \bar{A}\bar{x} \geq \bar{b} \) as follows

\[
\bar{f} = \bar{A}\bar{x} - \bar{b} = (\bar{A},-\bar{b}) \begin{bmatrix} \bar{x} \\ 1 \end{bmatrix} = H \begin{bmatrix} \bar{x} \\ 1 \end{bmatrix} \geq 0
\]

where

\[
H = (\bar{A},-\bar{b}) \quad \text{and} \quad \begin{bmatrix} \bar{x} \\ 1 \end{bmatrix} \geq 0
\]

The vector \( \bar{f} = H \begin{bmatrix} \bar{x} \\ 1 \end{bmatrix} \geq 0 \) is referred to as the feasibility vector of the original linear system of inequalities \( \bar{A}\bar{x} \geq \bar{b} \).

2.1. A Literature Overview and Motivation

Traditionally, linear inequalities are resolved as linear programs using either the Simplex methods[1][2][3] &[4], Crisis Cross algorithms[5], interior point method[6] projective algorithms[1][7], path-following algorithms[8],...
3. The Airspace Analyzer Application

The MITRE Corporation’s Center for Advanced Aviation Development (CAASD) has developed a tool, airspace-Analyzer (AANA) [13][14], which simulates automated air traffic control (ATC). It separates traffic while respecting aircraft performance and ATC restrictions and procedures (REFs). We test the LIS algorithms using LPs generated by airspace-Analyzer, which for the entire continental US airspace (3000 simultaneous aircraft) typically have several hundred thousand variables and about a million constraints.

At each tick of its simulation clock, airspace-Analyzer iterates the following steps:

(a) Resolve the scenario at the current simulation time, looking out into the future to some time horizon (typically six minutes).

(b) Advance the simulation clock by a fixed time-increment (typically two minutes)

(c) Incorporate any resolution maneuvers requiring changes in aircraft positions at the new simulation time.

The inputs are the planned (X, Y, Z) positions of each aircraft at future times, along with restrictions and airspace parameters. Airspace-Analyzer represents the (x, y, z) positions of each aircraft at future times as variables in a linear program (LP). These match the input (X, Y, Z) unless a maneuver is needed, e.g. for separation vs. another aircraft. The LP’s constraints represent ATC constraints—on aircraft performance, permissible altitudes and speeds, and sufficient vertical or horizontal separation for each pair of aircraft. Some of these are nonlinear, but can be approximated linearly (so that they are made more, not less, stringent). The LP’s objective function maximizes forward progress, less certain weighted penalties discussed below.

The LP is designed to always be feasible, even though there might not be enough airspace to meet all ATC restrictions. For instance, suppose Aircraft A should be at or above 25000’ t=5. This could be expressed as “za,5>=25000”. However, traffic might be so dense that safe separation is inconsistent with this constraint. To assure LP feasibility, this constraint (say, the i-th) is represented as: “za,5>=25000 - d_i”, where d_i is a new (nonnegative) LP variable. To drive the value of d_i to 0 when possible, a term C_i d_i is subtracted from the objective function. The value of the constant C_i reflects the importance of driving this d_i to 0, relative to other “d” variables for other constraints. Separation constraints would have the highest value for C (virtually infinite). The LP typically includes constraints specifying desirable but non-essential ATC features (e.g. “keep the maneuvers simple”), with C values lower than those specifying ATC restrictions.

The full airspace-Analyzer LP will be published in a future paper. A simplified form appears in[13]. An even simpler version suffices here; it provides speed and separation constraints for a two aircraft in the horizontal dimension only.
Aircraft are modeled to fly along trajectories, which are sequences of points (called cusps) in XYZ-space. Between consecutive cusps, aircraft are assumed to fly in piecewise linear segments; maneuvers (turns, speed changes, vertical maneuvers) occur only at cusps. A simplifying assumption is made that maneuvers are instantaneous, so that one may interpolate along such segments to find the aircraft's continuous positions in XYZ-space at times between two consecutive cusps.

![Figure 2](image)

Figure 2. Defining A Feasible Region Between Aircraft

Figure 2 shows an aircraft moving in direction \( \Psi \) (eastbound in the figure) between Cusp (i-1) and Cusp (i). It is modeled as being able to make an (instantaneous) turn right or left by a maximum angle of \( \Phi \) at Cusp (i-1). It then flies linearly at a speed that can range from minSpeed (inner circle) to maxSpeed (outer circle). We can define \( \Phi \) as:

\[
\Phi = \arccos \left( \frac{\text{minSpeed}}{\text{maxSpeed}} \right)
\]

The region between the two circles (possible positions for Cusp (i)) is nonlinear and nonconvex, but a useful subset of this can be bounded by a convex 5-sided polygon. The inequality for the long side of the polygon in Figure 2 is

\[
(x_{i} - x_{i-1}) \cos(\Psi + \Phi) + (y_{i} - y_{i-1}) \sin(\Psi + \Phi) \leq (\Delta T) \cos(\Phi/4) \maxSpeed,
\]

where:

\[
\Phi = (3\Phi/4, \Phi/4, -\Phi/4, -3\Phi/4) \text{ for } s = 1, 2, 3, 4.
\]

### 3.1. Separation Inequalities

To define (horizontal) separation inequalities, it is useful to consider the aircraft in relative rather than absolute space. Figure 3 (a) and (b) show an example 90° encounter between Aircraft \( j \) and \( k \), in absolute space and relative space, respectively. In Figure 3 (a), Aircraft \( j \) flies north, passing ahead and to the right of the eastbound Aircraft \( k \). In this example there are four segments (\( M = 4 \)). The nominal positions of Cusp (i, j) and Cusp (i, k) are shown for \( i = 0, 1, 2, 3, \) and 4, assuming \( j \) and \( k \) ignore each other and fly their preferred (straight) route. In Figure 3 (b), a coordinate transformation is applied, so that Aircraft \( j \)'s position is fixed at the origin (for \( i = 0, 1, 2, 3, \) and 4). Aircraft \( k \) is seen to be moving southeastward relative to \( j \). Note that the vector from \( j \) s i-th cusp to \( k \)'s i-th cusp, for all values of \( i \), is the same in both Figure 3 (a) and Figure 3 (b).

![Figure 3](image)

Figure 3. Two Aircraft Encounter and Their Separation
In Figure 3 (c), a circle is drawn around a specified position with radius equal to the required separation distance (denoted DREQ). Aircraft j and k are separated by at least DREQ if and only if k’s segments in relative space all stay outside the circle. The task here is to so specify, using only linear inequalities/equalities in the airspace-Analyzer variables.

Airspace-Analyzer’s means for asserting in its LP that the Figure 3 (c) aircraft are always safely separated is shown in Figure 3 (d). The four segments of k are labeled $S_i$, for $i = 1, 2, 3,$ and $4$. Points are shown on the circle. If for each $i$, $S_i$ is on the opposite (safe) side of the circle from a line tangent to the circle at then $S_i$ never enters the circle, so k never enters the circle, and is safely separated at all times from j.

To assure that the $i$-th segment of Aircraft k in relative space is on the proper (safe) side of the tangent line at $\theta_i$, it is necessary that the beginning and ending cusp are on the proper side. The following pair of linear inequalities is satisfied if and only if the two endpoints of segment $S_i$ are entirely on the safe side of the tangent line at $\theta_i$:

\[
\begin{align*}
(x_i - b) \cos \theta_i + (y_i - y_j) \sin \theta_i \geq & DREQ \quad \text{(for} \quad 2 \leq i \leq M) \\
(x_i - b) \cos \theta_i + (y_i - y_j) \sin \theta_i \geq & DREQ \quad \text{(for} \quad 1 \leq i \leq M)
\end{align*}
\]

3.2. Spacing Inequalities

Airspace-Analyzer also allows two aircraft ($j$ and $k$) to be given a miles-in-trail (MIT) constraint. That is, they are to be spaced a certain distance at a certain time by a certain distance. For instance, if $j$ is supposed to be 10 MIT ahead of $k$ in the eastbound direction at time $i$, the inequality is $x_{ij} - x_{ik} \geq 10$.

3.3. Additional Constraints Needed to Achieve Air Traffic Control Objectives

Many other constraints are used in airspace-Analyzer, to accomplish a variety of air traffic control goals. Examples include:

- Altitude Restrictions (e.g. an aircraft reaching a certain position should be at or below a specific altitude)
- Sector Boundaries (e.g. an aircraft should stay within its sector until its planned exit position)
- Simplicity (Suppose an aircraft’s trajectory has four segments (as in Figure 3). If the first two and the last two are collinear the maneuver is simpler than if not. It is possible to extract a small penalty for a solution that fails to make them collinear.}

3.4. One-aircraft and Two-aircraft Constraints

For the purposes of this paper, it is sufficient to know that speed constraints (from Figure 3-1) and the constraints of Section 3.3 reference only a single aircraft. The only constraints that refer to more than one aircraft are separation and spacing constraints; and they refer to exactly two aircraft. In our decomposition strategy, we refer to the subset of constraints that reference exactly one particular aircraft as the “atomic” LP for that aircraft. In 3D, separation is accomplished by separating trajectory segments from a cylinder or ellipsoid by tangent planes. Niedringhaus’s 1995 IEEE paper [13] provides additional details.

4. A Simplex Alike Method to Reduce

\[ |f| \]

In this section, we illustrate how an ordinary linear program (LP) may be converted to a self-dual Linear System Feasibility (LSF) problem such that a technique similar to the Simplex method used in linear programming is applied to reduce the worst infeasibilities for a selected small subset of the linear constraints of a given LSF.

4.1. Self-Dual LP as LSF

For a given linear program (LP), both the primal and dual LPs may be combined into a self-dual LP so that the dual is the same as the primal and the objective function becomes simply a pair of linear inequalities. Converting LPs into self-dual form, we can enforce homogeneity with one additional dimension as:

The primal LP:

\[ \text{Max} \{ c \cdot x \mid A \cdot x \leq b; l \leq x \leq u \} \]

The dual LP: \[ \text{Min} \{ b \cdot y \mid y \cdot A = c; 0 \leq y \} \]

Optimality: \[ c \cdot x = b \cdot y \]

The self-dual LP as LSF [15][[16]]:

\[
\begin{bmatrix}
0 & -A & b \\
A^T & 0 & -c \\
-b & c & 0 \\
I & 0 & 0 \\
0 & I & -l \\
0 & -I & u
\end{bmatrix}
\]

Note that solution to the self-dual LP as LSF implies both primal and dual feasibility of the original LP. In other words, a feasible self-dual LSF enforces optimality for the original LP with the primal objective equal to the dual objective, i.e., \[ c \cdot x = b \cdot y \].

Inequalities (2) is only one of many possible LSF for an LP, alternatively LSF formulation for LP with different primal and dual variable bounds do exist.

Applying row permutation to an LSF such that the feasibility vector, $f$, is sorted in descending order. In other words, rows in $s$ are regrouped into four distinct sub-matrices such that all the rows with positive feasibility ($f_i > 0$) are in $S_p$, all the rows with zero feasibility ($f_i = 0$) are in $S_o$, all the rows with negative feasibility ($f_i < 0$) are in $S_n$, and all the rows with zero feasibility ($f_i = 0$) are in $S_z$. The primal and dual feasibility of the original LP. In other words, a feasible self-dual LSF enforces optimality for the original LP with the primal objective equal to the dual objective, i.e., \[ c \cdot x = b \cdot y \].
all the rows with the worst infeasibility 

\( f_{-\infty} = \min \{ f_k \ | \ \forall f_k < 0 \} \), are in \( S_{-\infty} \), and the rest of the rows \( f_{-\infty} < f_h < 0 \) are in \( S_n \).

\[
f_{\downarrow} = P * f = P * S * t = \begin{bmatrix} f_p (> 0) \\ f_z (= 0) \\ f_n (< 0) \\ f_{-\infty} (\min) \end{bmatrix} = \begin{bmatrix} S_p \\ S_z \\ S_n \\ S_{-\infty} \end{bmatrix} * t \tag{3} \]

Note that all rows in \( S_{-\infty} \) have the same worst (or maximum) infeasibility \( f_{-\infty} \).

### 4.2. ASimplex alike Method for \( f_{-\infty} \)

The Simplex method is used to solve either the primal or the dual LP. To apply the Simplex method, one starts with the identity matrix as a basis matrix for all the column variables. A basis matrix with its rank equal to the total number of column variables is maintained by exchanging some or all of the basis columns and rows with non-basis rows or columns. For the LSF, the feasibility vector is sorted in descending order with respect to its feasibility. We apply a technique similar to the Simplex method only to constraints with the worst infeasibilities, \( S_{-\infty} \). The following recursive procedure is used to assure that the set of constraints with the worst infeasibilities is always full ranked and the set size of \( S_{-\infty} \) is kept as small as possible.

Note that the size and row indices for the worst infeasibility, \( S_{-\infty} \), changed from one step to the next, hence, the Simplex method is not directly applicable; however, we can use the same technique inverting a nonsingular basis matrix for the rows in \( S_{-\infty} \) to compute an adjustment to the vector \( t \), as \( \Delta t \) such that the infeasibility for all the rows in \( S_{-\infty} \) is reduced simultaneously and equally.

Sort feasibility, \( f_i \), for rows in \( S_{-\infty} \) as \( f_{\downarrow} \).

Let \( r = \text{rank of } S_{-\infty} \), \( k = \| S_{-\infty} \| \)

where \( |S| \) is the number of rows in \( S \).

While \((0 < k < r)\)

{ 

Replace set \( S_{-\infty} \) by the set of rows in \( S_{-\infty} \) with the same worst infeasibility.

Compute \( k \) and \( r \).

}

Note that the worst infeasibility for rows in \( S_{-\infty} \) are subjected to errors in rounding and truncations; gaps between rows may be scaled to magnify their difference. A LSF scaling algorithm is provided later in Section 9. Once we have a full-ranked \( S_{-\infty} \), the following steps are used to reduce the worst infeasibility \( f_{-\infty} \). The superscription, \( C \), is used to denote the complement of a sub matrix.

\[
\text{Let } S = \begin{bmatrix} C_{S} \\ S_{-\infty} \end{bmatrix} \tag{4} \]

and assume that \( S_{-\infty} \) is full ranked with \( r \) rows; i.e., we have

\[
\text{rank}(S_{-\infty}) = r \tag{5} \]

We can perform column permutations to rearrange columns of \( S_{-\infty} \) as:

\[
(T_{-\infty})_{r \times n} = S_{r \times n} * Q_{n \times n} = \begin{bmatrix} (B_{-\infty})_{r \times r} \\ (B_{-\infty})_{r \times (n-r)} \end{bmatrix} \tag{6} \]

where \( (B_{-\infty})_{r \times r} \) is a nonsingular square submatrix of \( (T_{-\infty})_{r \times n} \).

Any adjustments as shown in Figure 3 to \( f_{-\infty} \) may be treated as the results of multiplying \( \Delta t \) by \( (B_{-\infty})_{r \times r} \), i.e.,

\[
\Delta f_{-\infty} = B_{-\infty} * \Delta t_B \tag{7} \]

Hence, for any \( \Delta f_{-\infty} \), we have

\[
\Delta t_B = B^{-1}_{-\infty} \ast \Delta f_{-\infty} \tag{8} \]

Let \( \Delta f_{-\infty} \mid 1_r \) where \( 1_r = (1,1,\ldots,-1)^T_{1 \times r} \)

\[
\Delta t = \begin{bmatrix} (\Delta t_B)_{r \times 1} \\ 0_{(n-r) \times 1} \end{bmatrix} \tag{9} \]

\[
T_{-\infty} \ast \Delta t \mid 1_r \tag{10} \]

\[
T_{-\infty} \ast (t + \Delta t) = (f_{-\infty} + |f_{-\infty}| |1_r = (-) |f_{-\infty} + |f_{-\infty}| |1_r = 0, \ (11) \]

Although \( \Delta t \) does remove the worst infeasibility \( f_{-\infty} \) for all rows in \( S_{-\infty} \) as shown in Figure 3, its impact on other rows in \( S_{-\infty} \) cannot be ignored. For each row, \( S_i \) in \( S_{-\infty} \), let

\[
s_i \ast Q_{n \times n} = ((s_i)_B \hat{\vdots} (s_i)_{B^c}) \quad \text{where} \quad (s_i)_B \text{ is an } 1 \times r \text{ row vector and} \]

\[
(s_i)_{B^c} \text{ is an } 1 \times (n - r) \text{ row vector} \tag{12} \]

Applying \( \Delta t \) to \( f_i \) at row \( i \), we obtain

\[
\Delta f_i = [(s_i)_B \hat{\vdots} (s_i)_{B^c}] \ast \Delta t_B = \begin{bmatrix} (s_i)_B \cdot B^{-1} \ast |1_r \end{bmatrix} \tag{13} \]
Note that the necessary conditions for \( \Delta f_{-\infty} \) to be beneficial are \( 0 < \Delta f_{-\infty} \) and \( f_{-\infty} + \mu_i \Delta f_{-\infty} \leq f_i + \mu_i \Delta f_i \) with \( 0 < \mu_i \). Solving for the maximum possible \( \mu_i \), we have
\[
\mu_i = (f_i - f_{-\infty}) / (\Delta f_{-\infty} - \Delta f_i) = (f_i - f_{-\infty}) / |f_{-\infty}| (1 - (s_i)_B \cdot B^{-1} * 1_r) \quad (14)
\]

Note that from Figure 4, \( \mu_i \) is where \( f_{-\infty} \) meets \( f_i \) with \( \mu_i \Delta B \) applied.

From Figure 3, the row in \( S_{-\infty}^C \) with the smallest \( \mu_i \) is the maximum improvement \( f_{-\infty} \) can be achieved with
\[
\Delta t = \begin{bmatrix} \Delta t_B \\ 0 \end{bmatrix}.
\]
Hence,
\[
\mu = \text{Min}_{\forall s_i \in S_{-\infty}^C = S - S_{-\infty}} \mu_i \quad (15)
\]

In summary, this algorithm deals with all the rows in \( S_{-\infty} \). Similar to the Simplex method, a small adjustment, \( \Delta t \), to the tentative solution, \( t \), is computed from the full-rank basis for all the rows in \( S_{-\infty} \). All the rows in \( S_{-\infty} \) are decomposed into \( \begin{bmatrix} S_d \\ S_r \end{bmatrix} \) while \( S_r \) has the same row rank as that of \( S_{-\infty} \). Let \( B \) be a nonsingular square sub-matrix of \( S_r \) such that \( S_r = [B, N] \). The basis matrix, \( B \), may be constructed from \( S_r \) by sequentially selecting linearly independent columns. Alternatives to pick linearly independent columns from \( S_r \) as a basis matrix \( B \) does exist. Similar to the Simplex method, we compute the maximum allowable adjustment to \( \Delta t \) as follows:
\[
t_0 = 0, \quad t_{k+1} = t_k + \Delta t_k \quad \text{while} \quad \Delta t_k = \mu |f_{-\infty}| B^{-1} * 1_r \quad \text{with} \quad \mu = \text{Min}_{\forall s_i \in S_{-\infty} = S - S_{-\infty}} \{(f_i - f_{-\infty}) / |f_{-\infty}| (1 - (s_i)_B \cdot B^{-1} * 1_r) \}, \quad (s_i)_B = (s_i)_1 \}
\]

Comparing to the Simplex method that operates on a full size nonsingular basis for the constraint matrix, \( A \), this Simplex alike basis matrix, \( B \), is only a very small subset computed from the linearly independent rows in \( S_{-\infty} \). In summary, the following three steps are applied recursively to reduce the maximum infeasibility, \( f_{-\infty} \).

**Step 1.** Assume that rank \( S_{-\infty} = |S_{-\infty}| = k \), select \( k \) linearly independent columns from \( S_{-\infty} \) such that \( S_{-\infty} = [B, N] \) where \( B \) is a square non-singular sub matrix of \( S_{-\infty} \).

**Step 2.** Compute the maximum possible reduction of the worst infeasibility \( f_{-\infty} \) for \( S_{-\infty} \) as
\[
\mu |f_{-\infty}| \quad \text{with} \quad 0 < \mu = \text{Min}_{\forall s_i \in S_{-\infty} = S - S_{-\infty}} \{(f_i - f_{-\infty}) / |f_{-\infty}| (1 - (s_i)_B \cdot B^{-1} * 1_r) \}
\]

**Step 3.** Adjust the vector \( t \) with \( t_0 = 0, \ t_{k+1} = t_k + \Delta t_k \), \( \Delta t_k = \mu |f_{-\infty}| B^{-1} * 1_r \)

Steps 1 to 3 to reduce \( |f_{-\infty}| \) may be applied recursively.

Figure 4 illustrates the maximum possible reduction of the worst infeasibility using this Simplex alike method. Note that the basis matrix is obtained from a small subset of all the rows (with the maximum infeasibility) and that its rank varies from one step to the next as the landscape of the feasibility vector, \( \vec{f} \) changed from one step to the next.

Figure 4 also illustrates the fact that utilizes the basis matrix for rows in \( S_{-\infty} \) to reduce simultaneously the value of \( |f_{-\infty}| \) may be bounded by a potentially blocking row, \( b \), with its feasibility \( f_b \) such that \( f_{b} + \Delta f_{b} = f_{b} + S_{b} * \Delta t_{b} = f_{-\infty} + B * \Delta t_{b} = f_{-\infty} + \Delta f_{-\infty} \) with \( \Delta f_{-\infty} = B * \Delta t_{b} = \mu |f_{-\infty}| * 1_r \).

5. Zero-crossing in LSF

The second algorithm utilizes nonzero coefficients of a given LIS to identify zero-crossing points for the LIS such that the sum of all infeasibilities may be reduced. Note that reducing the sum of all infeasibilities may or may not result in the reduction of the maximum infeasibility (\( f_{-\infty} \)). For an infeasible row (such as row \( i \)) to become feasible, one must adjust the feasibility \( f_i \) through a nonzero coefficient, \( s_i \), of a related column, \( j \), of row of \( S_i \). Such nonzero column of \( S_i \) defines the set of all-zero crossings for \( f_i \). This
algorithm identifies all the zero-crossing points for $S$ and tests their impacts on both the sum and maximum infeasibility for $S$. Among all possible zero-crossing points, one can identify columns that can be used to maximally reduce the sum of infeasibilities over all the infeasible rows. Simply stated, this algorithm computes the difference in the sum of infeasibility for every nonzero column coefficient at its zero-crossing points for any infeasible row. Each nonzero coefficient associated with an infeasible row uniquely defines such a zero-crossing opportunity. Let

$$I_1 = \sum_{\forall f_i < 0} f_i$$

be the sum of all infeasible rows of $f$, any nonzero coefficient, $s_{iu} \in S$ at row $i$ column $u$ provides a zero crossing point for an infeasible row, $f_i$, with adjustment of column variable, $t_u$, such that,

$$g_i = f_i + \Delta t_u s_{iu} = 0.\text{ Such a zero crossing point for } f_i \text{ will have its global implication on the new sum of all infeasible rows. In other words, the new infeasibility sum will be } I_2 = \sum_{\forall g_j < 0} g_j \text{. Consequently, we will have a net gain or loss of the infeasibility sum for the feasibility vector, } f, \text{ as } \Delta I = I_2 - I_1 \text{. i.e.,}$$

$$\Delta I = I_2 - I_1 = \sum_{\forall g_j < 0} g_j - \sum_{\forall f_i < 0} f_i$$

with $g_i = f_i + \Delta t_u s_{iu}$ where $\Delta t_u = t_{iu} = -f_i / s_{iu}$

(16)

Figure 5. Reducing the Sum of Infeasibility with Zero-Crossings

The maximum number of possible zero-crossing points is equal to the total number of nonzero coefficients in $S$. One can easily identify a zero crossing point that yields the maximum $\Delta I$.

This algorithm may be applied recursively to reduce the sum of all infeasibilities of $S$ or the size of $S_{-\infty}$ or both for a non-zero column coefficient, $s_{iu}$ or $s_{iu}$.

6. De-grouping the Set of Rows with the Worst Infeasibility

This Simplex alike method is capable of recursively reducing the magnitude of the maximum infeasibility, $|f_{-\infty}|$. Applying the zero-crossing for infeasible rows is capable of recursively reducing the sum of infeasibilities for all rows. It is likely that the set size of $S_{-\infty}$ may increase noticeably. An algorithm is needed to reduce the set size of $S_{-\infty}$. This is accomplished with columns that are reachable from rows in $S_{-\infty}$. If a column, $j$, reachable (with non-zero coefficient, $s_{ij}$) by the rows in $S_{-\infty}$ has row coefficients for those rows are of the same sign (either positive or negative), then one can reduce the set size of $S_{-\infty}$ by moving up those rows from the set of $S_{-\infty}$ since the infeasibility of all those rows may be reduced concurrently by adjusting the column variable in the beneficial direction, i.e., positive or negative, as shown in Figure 6. Simply stated, the following steps are identified to recursively reduce the number of rows with the worst infeasibility, $f_{-\infty}$, in any LIS:

**Step 1:** Determine the set of all rows with the worst infeasibility $f_{-\infty} : S_{-\infty}$

**Step 2:** Determine the columns that are reachable from any rows in $S_{-\infty} : C_{-\infty}$

**Step 3:** For each $c_i$ in $C_{-\infty}$ determine all the rows in
that are reachable from \( c_i : R_{C_x} \)

**Step 4:** Check if all rows in \( R_{C_x} \) are of the same sign (positive or negative)

**Step 5:** If \( R_{C_x} \) is of single sign, the size of \( S_{-\infty} \) can be reduced by the size of \( R_{C_x} \).

In other words: adjustment of variable for column \( c_i \) will reduce the size of \( S_{-\infty} \) by the size of \( R_{C_x} \)

\[
| S_{-\infty} | - | R_{C_x} |
\]

7. **Convergence/Completeness Analysis**

The convergence of the proposed LIS algorithms, namely, the Simplexalike method to reduce \( f_{-\infty} \), zero-crossing to reduce sum of infeasibility, and de-grouping to reduce the set size of rows in \( S_{-\infty} \) may be illustrated with the following arguments:

Let \( t \) be a feasible solution for \( f - S \cdot t \geq 0 \) if such feasible solution does exist, and \( x \) be any vector such that \( g - S \cdot x \geq 0 \). Sort the vector \( g \) in descending order as

\[
g_+ = P \cdot S \cdot x = P \cdot g = \begin{bmatrix} g_p (\geq 0) \\ g_n (< 0) \\ g_{-\infty} (Min) \end{bmatrix} = \begin{bmatrix} G_p \\ G_n \\ G_{-\infty} \end{bmatrix} \cdot x,
\]

\[
S \cdot t = S \cdot (x + (t - x)) \geq 0
\]  

(17)

Note that we have \( S_{-\infty} \cdot t \geq 0 \), there are only three cases:

**Case 1:** \( S_{-\infty} \cdot t > 0 \) with \( \| S_{-\infty} \| \geq \| G_{-\infty} \| \)

**Case 2:** \( S_{-\infty} \cdot t > 0 \) with \( \| S_{-\infty} \| < \| G_{-\infty} \| \) and

**Case 3:** \( S_{-\infty} \cdot t = 0 \)

For Case 1, the vector \( (t-x) \) reduces \( \| g_{-\infty} \| \) for all rows in \( G_{-\infty} \). Hence, the economical Simplex method has nonzero solution.

For Case 2, \( \| g_{-\infty} \| \) is unchanged, however, \( \| G_{-\infty} \| \) is reduced by the vector \( (t-x) \). Hence, de-grouping has no trivial solution.

For Case 3, we have \( S_{n} \cdot t \geq 0 \), hence the vector \( (t-x) \) must reduce the sum of infeasibilities for \( g \). Since all adjustments to \( x \) for \( g \) are the zero-crossing points, at least one such point must reduce the sum of infeasibilities in \( g \). Hence, zero-crossing will have nonzero solution.

Consequently, we can always find an adjustment to \( x \) as \( \Delta x \) from \( g \) and \( t \) such that \( S(x + \Delta x) = g + \Delta g \) has either a smaller \( \| g_{-\infty} \| \) (larger minimum \( g \)), or a smaller \( \| G_{-\infty} \| \), or a smaller sum of infeasibilities.

8. **Decomposition, Combination, and Scalability of LIS**

It is possible to decompose a given LIS into two coupled LISs with row and column permutations. The children LISs can then be solved in parallel. The solution of the parent LIS is then obtained with minimum changes to the combined solutions of the children LISs and the combined column variables. Figures 7 and 8 illustrate this LIS decomposition and recombination relation that can easily be implemented in any parallel computing platform.

9. **LIS Scalability**

The following analysis illustrates the scalability of LIS with respect to the solution vector. Scalability can be utilized to separate true infeasibility gaps from the errors of rounding and truncation. Consequently, scaling LIS can result in higher precision and fast convergence.

Assume that one wishes to solve
\[ f = Cx - b = [C,-b] \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0 \text{ at } |f_{\rightarrow}| \leq 10^{-5} \]

If we solve \( g = Cy - 10b = [C,-10b] \begin{bmatrix} y \\ 1 \end{bmatrix} \geq 0 \text{ at } |g_{\rightarrow}| \leq 10^{-4} \), we have

\[ g / 10 = [C(y/10) - b] = [C,-b] \begin{bmatrix} y/10 \\ 1 \end{bmatrix} = f = [C,-b] \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0 \text{ at } |f_{\rightarrow}| \leq 10^{-5} \]

Consequently, achieving worst infeasibility at \( 10^{-4} \) for \( g \) is the same as achieving \( 10^{-5} \) for \( f \). The scaling of the vector \( b \) has the effect of magnifying the infeasibility gaps with the same scale. A very large scaling factor for \( b \) can magnify the infeasibility by the same scale and hence decoupling ordinary rounding and truncation errors from the true infeasibility gaps. Experiments with Java code has confirmed this observation and significant reduction in run time was also observed with very large LIS scaling factor of \( 10^6 \) for CAASD’s airspace-Analyzer application.

10. Advantages and Challenges of LIS in LSF Form

In summary, the authors have shown that converting ordinary system of linear inequalities to its feasibility form, LSF, offers the following advantages:

- LSF may be solved with three new algorithms that recursively reduce the sum of infeasibility of all constraint inequalities, the maximum infeasibility among all the linear constraints, and the number of constraints that share the same worst case infeasibility. Each algorithm is shown to utilize CPU time that is proportional to the number of nonzero coefficients (NNZ) of a small subset of all the nonzero coefficients of the constraint matrix, \( A \), that defines the system of linear inequalities.

- LSF can be solved with recursive binary decomposition and recombination for very large number of variables and constraints in the range of tens of thousands.

- LSF covers all linear systems including system of linear equalities (singular or non-square systems included), linear programs formulated in self-dual form, eigenvector program, bounded linear systems, orthogonal vectors, and ordinary system of linear inequalities.

- LSF provides a simple scaling capability that can be used to improve the precision and speed of its key algorithms.

- LSF can be easily implemented in a parallel computing platform to take advantage of concurrent CPU, multi-threading, and the advantage of Graphics Processing Unit (GPU) computing.

Initial tests with hundreds of large size sparse LPs from scenarios generated by CAASD airspace-Analyzer (aA) clearly demonstrated these advantages. As shown in Figure 7, LIS can recursively obtain a feasible solution of a given system of linear inequalities in LSF form from the solutions of its components as a left and a right child LSF with only minor changes to the combined solutions such that to remove infeasibility that is purely the results of additional nonzero coefficients in both the joint rows and joint columns.

Hence, one would expect that LIS will be very fast for applications that are very sparse with small size of joint rows or columns for the parent node of its left and right child nodes. We have coined such applications as “LP local” to reflect the following two properties: Its sparsity increases with size and its optimal cut set of its left and right children has a small number of joint rows or columns. CAASD’s UTest cases as 13 level decomposed binary trees shows that GLPK has a timing profile slightly better than the \( O(n^{2.5}) \) where \( n \) is the number of variables (columns) and LIS without threading, GPU, or parallelization has a timing profile that is slightly worse than \( O(\rho * m * n) \) where \( m \) is the number of constraints (rows) and \( \rho = \text{NNZ}/(m * n) \) is the sparsity of the constraint matrix. A. Using MITRE CAASD application airspace-Analyzer as a linear program (LP) for conflicts resolution and operational constraints for the entire continental US with 1/4 million variables, 1/2 million constraints, and 2.3 million nonzero coefficients (NNZ) of the LSF. Initial experimental timing profiles using both LIS and GLPK for the entire continental UTest cases as 13 level decomposed binary trees shows that GLPK has a timing profile slightly better than the \( O(n^{2.5}) \).
timing comparison as shown in Figure 10. The x-axis on Figure 9 is the level index of the parent nodes on such a 13 level binary tree. In Figure 9, the recorded CPU time from LIS and GLPK for each LP as a node on this binary tree are compared with the sparsity of the matrix, A, for each node and the anticipated computational complexity of $O[NNZ] \approx k \cdot NNZ$ for LIS and $O[n^{2.5}] \approx h \cdot n^{2.5}$ for GLPK. The values for the multipliers $k$ and $h$ were determined experimentally with interpolation to available data points. Note that there are missing points for GLPK as it has exceeded the CAASD 64-bit Linux computer system’s maximum RAM (40 GB) configured. Highlight of Figure 9 may be summarized as follows:

- The sparsity, $\rho = \frac{NNZ}{(m \cdot n)}$, decreases noticeably for nodes on this 13 level tree from the atom leaves at level 0 to the root node at level 13.
- LIS Timing were slightly worse than that of $h \cdot n^{2.5}$
- LIS Timing were slightly worse than

$$k \cdot NNZ \approx k \cdot \rho \cdot m \cdot n \text{ with } \rho = \text{sparsity}$$

Note that the binary decomposition strategy is uniquely suitable for very sparse matrix such that only neighboring variables are strongly connected with very little joint rows or columns for variables that are remotely related. CAASD aA applications clearly demonstrate this particular LP-local characteristics. The traditional Bender decomposition with row generation and Dantzig-Wolfe decomposition[3][4] with column generation techniques that do require specific topological structure of the constraint matrix are not directly applicable to Figure 8 as a nesting binary tree.

There are challenges that may be overcome later with additional insight and intelligent fine tuning of the key algorithms to resolve outstanding questions regarding its applicability to integer programming, binary programming, parallel computing enhancements, and the ultimate resolution of the worst case computation complexity for the entire class of NPC problems[10].

![A Comparison of LIS with GLPK Timing vs. size](image-url)

**Figure 9. Preliminary LIS Timing Profile With aA Applications**
11. Conclusions and Future Work

In conclusion, the authors present innovative concepts in dealing with very large system of linear inequalities with number of variables and/or constraints close to or over millions. First, all linear systems (equalities and inequalities included) are converted to its homogeneous form such that the right hand side are all zeros. This is accomplished by increasing the number of columns of the constraint matrix by one and replacing the solution vector, $x$, by $t$ where

$$t = \begin{bmatrix} x \\ 1 \end{bmatrix}.$$  

With linear systems in homogeneous form, linear inequality or equality problems become a searching for a column vector, $t$, with a constraint matrix, $S$, such that $f = St \geq 0$. Consequently, the vector, $f$, is a measurement of linear feasibility of the solution vector $t$ with respect to the constraints vectors as rows in $S$. We also demonstrate that if a homogeneous linear inequality system, $f = St \geq 0$ does have a solution with a given initial $St_0 \geq 0$, i.e., not all $St_0 \geq 0$, then one must be able to reduce either the sum of infeasibility computed as the sum of all negative components of the feasibility vector, $f_0 = St_0$, or the worst case infeasibility as the minimum component of $f_0$ ($<0$), or the number of rows in $S$ with such a worst case infeasibility. With such insights, we designed three key algorithms aiming to reduce the sum, the worst case, or the number of rows in $S$ with the worst case infeasibility to recursively move from any initial vector, $f_0$ to a feasible solution such that $f = St \geq 0$. Such three key algorithms are a Simplex alike algorithm aiming to reduce the worst case infeasibility in $f$, a zero-crossing algorithm aiming to reduce the sum of all infeasible rows in $S$ (as revealed by the column vector $f = St$), and a row de-grouping algorithm that is capable of separating linearly dependent rows in $S$ such that all the rows in $S$ with the worst case infeasibility are linearly independent. Combining this set of three key algorithms, it is demonstrated that with a computational complexity slightly worse then the total number of nonzero elements in the constraints matrix, $S$, one can locate a solution for the given linear inequality system. This is verified with linear programs (LPs) in self-dual form such that both the prime and the dual systems are explicitly included and a solution exists. These three new algorithms were coded and tested in Java code with the CAASD aviation application, a $A$, that handles number of variables and/or constraints over one million in a large number of test cases. Experimental test data does show that the answers obtained by such a CAASD Linear Inequalities Solver (LIS), matched exactly solutions obtained from either the Cplex by ILOG, or that obtained with GLPK as the Gnu LP Solver. A preliminary comparison of the CPU time with increasing size in number of variables and constraints are provided. In most cases, the advantage of numerical stability, CPU time vs. problem size, and scalability as described by this paper are demonstrated. Note that such an approach is also applicable to system of linear equalities and Eigen vectors solutions.

Major findings of this research are: first, the Simplex alike approach is more economical than the original Simplex as it only applies to rows in $S$ with the worst case infeasibility (minimum value $f_i$ of the feasibility vector, $f = St$); hence, it is much faster and more productive in reducing the worst case infeasibility; second, zero-crossing provides a great number of linear paths to reduce the sum of total infeasibility; and third, row de-grouping is the most efficient way to identify linearly independent rows for the group of rows in $S$ with identical worst case infeasibility for a given column vector, $t$; fourth, homogeneous linear systems may be scaled to either speed up or increase the precision desired for such an innovative LIS approach; lastly, at high precision, row infeasibility and rounding errors becomes indistinguishable and algorithms can be properly terminated. Note that linear systems in homogeneous form with both equalities and inequalities included does remove the distinction between equalities and inequalities, hence, LIS can be used to solve system of linear equalities just as it solves linear inequalities as $f = St = 0$ becomes

$$\begin{bmatrix} S \\ -S \end{bmatrix} \preceq \begin{bmatrix} 0 \\ 0 \end{bmatrix}.\text{Namely, } St = 0 \iff \begin{bmatrix} S \\ -S \end{bmatrix} \preceq t \geq 0.$$

Furthermore, homogeneous linear systems (equalities and inequalities included) provide a handy normalization process such that rows of the matrix $S$ are all unit vectors and the normalized solution, $t_u = t/\|t\|$ is also a unit vector. Note that unit vectors are numerical stable free from numerical overflow or ill-conditioned problems. Future works include a generalization of the Gaussian Elimination (GGE) for linear systems (equalities and inequalities included) to identify the feasible interval (null, unique, or a real interval with an upper bound and a lower bound). Another piece of our innovative idea for the homogeneous linear systems is to locate a solution or solutions of a homogeneous linear system on the surface of an $n$-dimensional hyper unit sphere with a radius of 1.0 known as the unit shell (US) with linear subspace projection and the concepts of equal distanced points on the unit shell. Such new ideas are currently under testing by the authors using the Excel tool and java code over very large linear systems with both equalities and inequalities included. The validity of such new innovative ideas is still being pursued and verified. Details of such innovative approaches closely related to the homogeneous linear systems solvers are beyond the scope of this paper. Hopefully, draft papers will soon be available for peer review shortly in additional to this paper on LIS.

ACKNOWLEDGMENTS

The authors acknowledge the helpful review, correction, and editorial advice applied to this draft paper by many of...
colleagues at MITRE, in particular, Dr. Glenn Roberts, David Hamrick, Dr. Leone Monticone, Dr. Leonard Wojcik, Matt Olson, and Jon Drexler. Our thanks also extended to key CAASD IT Resource Center (ITRC) staff provided the resource, tools, and software support in both conventional and parallel computing. Valuable advice and feedback from SAP reviewers and editors are also greatly appreciated.

REFERENCES


